LING 408/508: Programming for Linguists

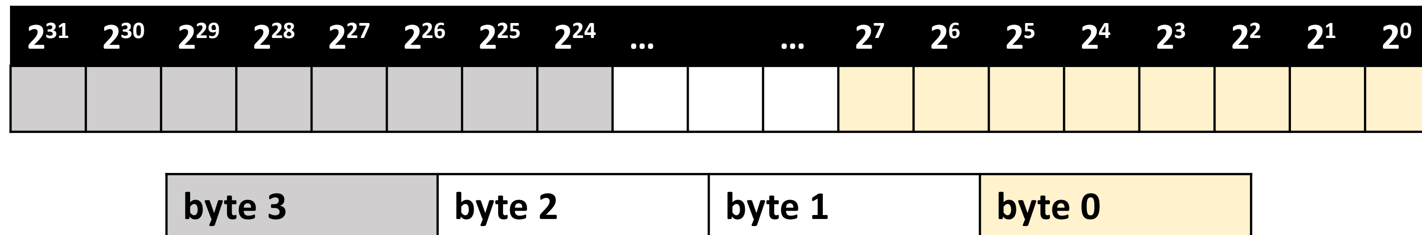
Lecture 1

Today's Class

- Ungraded homework exercise review
- 32 bit floating point demonstration
- Homework 1 (due tomorrow midnight)

Ungraded Homework Exercise Review

- What would the integer representation of the speed of light (in m/s) look like in binary representation as a 32 bit number?
- *c* = 299,792,458



Ungraded Homework Exercise Review
Homework Exercise

- What would the integer representation of the speed of light (in m/s) look like in binary representation as a 32 bit number?

n	2^n	n	2^n	n	2^n
0	1	11	2,048	22	4,194,304
1	2	12	4,096	23	8,388,608
2	4	13	8,192	24	16,777,216
3	8	14	16,384	25	33,554,432
4	16	15	32,768	26	67,108,864
5	32	16	65,536	27	134,217,728
6	64	17	131,072	28	268,435,456
7	128	18	262,144	29	536,870,912
8	256	19	524,288	30	1,073,741,824
9	512	20	1,048,576	31	2,147,483,648
10	1,024	21	2,097,152	32	4,254,967,296

$c =$ 299,792,458

2^{28} = 268,435,456
left with 31,357,002

2^{24} = 16,777,216
left with 14,579,786

2^{23} = 8,388,608
left with 6,191,178

2^{22} = 4,194,304
left with 1,996,874

2^{20} = 1,048,576
left with 948,298

2^{19} = 524,288
left with 424,010

2^{18} = 262,144
left with 161,866

2^{17} = 131,072
left with 30,794

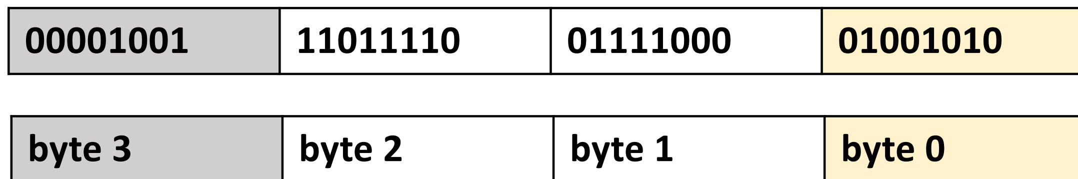
2^{14} = 16,384
left with 14,410

2^{13} = 8,192
left with 6,218

2^{12} = 4,096
left with 2,122

Ungraded Homework Exercise Review

- What would the integer representation of the speed of light (in m/s) look like in binary representation as a 32 bit number?



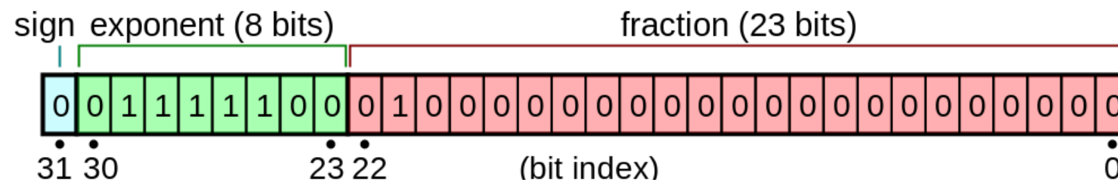
n	2^n	n	2^n	n	2^n
0	1	11	2,048	22	4,194,304
1	2	12	4,096	23	8,388,608
2	4	13	8,192	24	16,777,216
3	8	14	16,384	25	33,554,432
4	16	15	32,768	26	67,108,864
5	32	16	65,536	27	134,217,728
6	64	17	131,072	28	268,435,456
7	128	18	262,144	29	536,870,912
8	256	19	524,288	30	1,073,741,824
9	512	20	1,048,576	31	2,147,483,648
10	1,024	21	2,097,152	32	4,254,967,296

$c =$ 299,792,458

$2^{11} = 2{,}048$	$2^6 = 64$	$2^3 = 8$	$2^1 = 2$
left with 74	left with 10	left with 2	left with 0

Last time

- Recall the speed of light:
  - $c = 2.99792458 \times 10^8$ (m/s)
- The 32 bit floating point representation (`float`) – sometimes called single precision - is composed of 1 bit sign, 8 bits exponent (unsigned with bias $2^{(8-1)}-1$), and 23 bits coefficient (24 bits effective).



- Can it represent *c* without loss of precision?
  - $2^{24}-1 = 16{,}777{,}215$
  - Nope. Let's demonstrate that!

32bitfloat.xlsx

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG
1	Sign	Exponent								Fraction																							
2	0	0	1	1	1	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	31	30	29	28	27	26	25	24	23		22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
4										binary point @ left of bit 22																							
5	Sval	Exponent Value (bits 30 to 23)								Fraction Value (1 + bits 22 to 0)																							
6	1	-3								1.25																							
7	Decimal Value																																
8	0.15625																																
9	ExpVal	Exponent																															
10	**-3**	0	1	1	1	1	1	0	0																								
11	**-2**	0	1	1	1	1	1	0	1																								
12	**-1**	0	1	1	1	1	1	1	0																								
13	**0**	0	1	1	1	1	1	1	1	bias is 127																							
14	**1**	1	0	0	0	0	0	0	0																								
15	**2**	1	0	0	0	0	0	0	1																								
16	**3**	1	0	0	0	0	0	1	0																								

$c = 2.99792458 \times 10^8$ (m/s)

Representing zero

- Let's not worry about representing zero in the spreadsheet…

In IEEE 754 binary **floating point** numbers, **zero** values are **represented** by the biased exponent and significand both being **zero**. Negative **zero** has the sign bit set to one.

Signed zero - Wikipedia
https://en.wikipedia.org/wiki/Signed_zero

Homework 1

math.pi in Python 3 reports the decimal value of PI to the best of its ability

```
ling538-20$ python3
Python 3.8.3 (v3.8.3:6f8c8320e9, May 13 2020, 16:29:34)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import math
>>> math.pi
3.141592653589793
>>>
```

Homework 1

- How close can you get to the Python value of PI given the 32 bit floating point in the spreadsheet 32bitfloat.xlsx?
- Does Python use a 32 bit floating point representation? Explain.
- Email: your answer to sandiway@email.arizona.edu
- SUBJECT: 408/508 Homework 1 YOUR NAME
- PDF file please only
- Submission should include a screen snapshot of your final spreadsheet
- Due date: tomorrow midnight
- Homework review: next time (Thursday)

Representing zero

- Let's not worry about representing zero in the spreadsheet...

In IEEE 754 binary **floating point** numbers, **zero** values are **represented** by the biased exponent and significand both being **zero**. Negative **zero** has the sign bit set to one.

Signed zero - Wikipedia
https://en.wikipedia.org/wiki/Signed_zero

Introduction: data types

- How about letters, punctuation, etc.?
- ASCII
  - American Standard Code for Information Interchange
  - Based on English alphabet (upper and lower case) + space + digits + punctuation + control (Teletype Model 33)
  - **Question**: how many bits do we need?
  - 7 bits + 1 bit parity
  - Remember everything is in binary …

C:
char

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Teletype Model 33 ASR
Teleprinter (Wikipedia)

Introduction: data types

order is important in sorting!

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

0-9: there's a connection with BCD. **Notice:** code 30 (hex) through 39 (hex)

Introduction: data types

- Parity bit:
  - transmission can be noisy
  - parity bit can be added to ASCII code
  - can spot single bit transmission errors
  - even/odd parity:
    - receiver understands each byte should be even/odd
  - Example:
    - 0 (zero) is ASCII 30 (hex) = 011000
    - even parity: 0011000, odd parity: 1011000
  - Checking parity:
    - Exclusive or (XOR): basic machine instruction
      - A xor B true if either A or B true but not both
  - Example:
    - (even parity 0) 0011000 xor bit by bit
    - 0 xor 0 = 0 xor 1 = 1 xor 1 = 0 xor 0 = 0 xor 0 = 0 xor 0 = 0 xor 0 = 0

x86 assemby language:

1. PF: even parity flag set by arithmetic ops.
2. TEST: AND (don't store result), sets PF
3. JP: jump if PF set

Example:

```
MOV al,<char>
TEST al, al
JP <location if even>
<go here if odd>
```

Introduction: data types

- UTF-8
  - standard in the post-ASCII world
  - backwards compatible with ASCII
  - (*previously, different languages had multi-byte character sets that clashed*)
  - Universal Character Set (UCS) Transformation Format 8-bits

Bits of code point	First code point	Last code point	Bytes in sequence	Byte 1	Byte 2	Byte 3	Byte 4
7	U+0000	U+007F	1	0xxxxxxx			
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx		
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx	
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

(Wikipedia)

Introduction: data types

Bits of code point	First code point	Last code point	Bytes in sequence	Byte 1	Byte 2	Byte 3	Byte 4
7	U+0000	U+007F	1	0xxxxxxx			
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx		
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx	
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

- Example:
  - あ Hiragana letter A: UTF-8: E38182
  - Byte 1: E = 1110, 3 = 0011
  - Byte 2: 8 = 1000, 1 = 0001
  - Byte 3: 8 = 1000, 2 = 0010
  - い Hiragana letter I: UTF-8: E38184

Shift-JIS (Hex):
あ: 82A0
い: 82A2

Many Windows programs (including Windows Notepad) add the bytes 0xEF, 0xBB, 0xBF at the start of any document saved as UTF-8. This is the UTF-8 encoding of the Unicode byte order mark (BOM), and is commonly referred to as a UTF-8 BOM, even though it is not relevant to byte order. A BOM can also appear if another encoding with a BOM is translated to UTF-8 without stripping it. Software that is not aware of multibyte encodings will display the BOM as three strange characters (e.g. "ï»¿" in software interpreting the document as ISO 8859-1 or Windows-1252) at the start of the document.

Introduction: data types

- How can you tell what encoding your file is using?
- Detecting UTF-8
  - Microsoft:
    - 1st three bytes in the file is EF BB BF
    - (*not all software understands this; not everybody uses it*)
  - HTML:
    - `<meta http-equiv="Content-Type" content="text/html;charset=UTF-8">`
    - (*not always present*)
  - Analyze the file:
    - Find non-valid UTF-8 sequences: if found, not UTF-8…
    - Interesting paper:
      - http://www-archive.mozilla.org/projects/intl/UniversalCharsetDetection.html